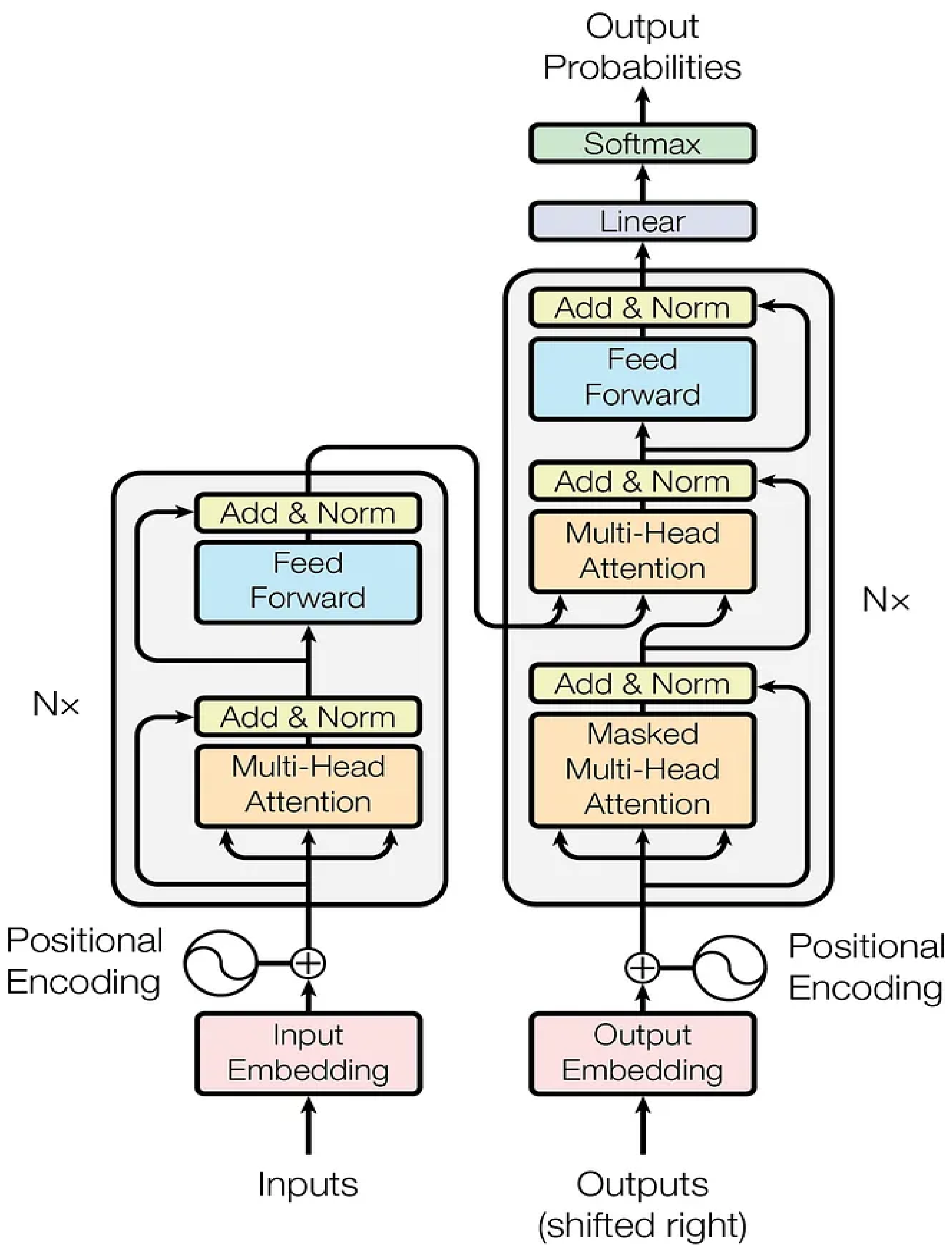
Transformer



Attention mechanisms: allow the model to focus on specific parts of the input sequence most relevant to the current prediction. Attending the informative parts of the sequence, even if they are far away, can help the model learn long-term dependencies.

Self-attention: self-attention allows the model to focus (weigh) on **different parts** of the input when processing each token, enabling it to capture complex relationships within the text.

Transformer Architecture: Key Components

1. Input Embedding + Positional Encoding

- Input Embedding: Converts input tokens into dense vectors. (ইনপুটকে টোকেনকে কনভার্ট করবে dense vectors এ।)
- Positional Encoding: Injects information about the position of each token (since there is no recurrence or convolution). (প্রতিটা টোকেনের পজিশনাল ইনফরমেশন পরবর্তীতে Injects করবে।

2. Encoder (Left Block)

Each encoder layer (repeated N times) has:

- **Multi-Head Self-Attention:** Lets each token attend to all tokens (including itself).
- **Add & Normalization:** Residual connection followed by Layer Normalization.
- **Feed Forward:** Fully connected layer applied to each position.
- **Another Add & Normalization:** For the FFN output.

The encoder outputs a sequence of vectors (contextual embeddings) of the same length as the input.

3. Decoder (Right Block)

Each decoder layer (also repeated N times) has:

- **Masked Multi-Head Self-Attention:** Ensures no "peeking ahead" during training.
- **Add & Norm**
- **Multi-Head Attention over Encoder Output**: Cross-attention to focus on encoder's output.
- **Add & Norm**
- **Feed Forward + Add & Norm**

The decoder uses both its own context and encoder output to generate the final output.

4. Output

After the decoder, a Linear layer and Softmax are applied to predict the next word in the sequence.

1. Encoder

The primary function of the encoder is to create a high-dimensional representation of the input sequence that the decoder can use to generate the output. Encoder consists multiple layers and each layer is composed of two main sub-layers:

Self-Attention Mechanism: This sub-layer allows the encoder to weigh the importance of different parts of the input sequence differently to capture dependencies regardless of their distance within the sequence.

Feed-Forward Neural Network: This sub-layer consists of two linear transformations with a ReLU activation in between. It processes the output of the self-attention mechanism to generate a refined representation.

Layer normalization and residual connections are used around each of these sub-layers to ensure stability and improve convergence during training.

2. Decoder

Decoder in transformer also consists of multiple identical layers. Its primary function is to generate the output sequence based on the representations provided by the encoder and the previously generated tokens of the output.

Each decoder layer consists of three main sub-layers:

Masked Self-Attention Mechanism: Similar to the encoder's self-attention mechanism but with a mask to prevent the decoder from attending to future tokens in the output sequence.

Encoder-Decoder Attention Mechanism: This sub-layer allows the decoder to focus on relevant parts of the encoder's output representation, facilitating the generation of coherent and contextually appropriate output sequences.

Feed-Forward Neural Network: This sub-layer processes the combined output of the masked self-attention and encoder-decoder attention mechanisms.

How Transformers Work

1. Input Representation

The first step in processing input data involves converting raw text into a format that the transformer model can understand. This involves tokenization and embedding.

ইনপুট ডেটা processing এর প্রথম ধাপ হল raw text কে এমন একটি format এ convert করা যা ট্রান্সফরমার মডেল বুঝতে পারে। এর মধ্যে রয়েছে tokenization and embedding.

Tokenization: Tokenization is the process of breaking down text into smaller, meaningful units called tokens. tokens can be words, subwords, or even character. Tokenization ensures that the text is broken down into manageable pieces.

Embedding: Then Each token is converted into a fixed-size vector using an embedding layer. This layer maps each token to a dense vector representation that captures its semantic meaning.

Positional encodings are added to these embeddings to provide information about the token positions within the sequence.

2. Encoder Process in Transformers

Input Embedding: The input sequence is tokenized and converted into embeddings with positional encodings added.

Self-Attention Mechanism: Each token in the input sequence attends to every other token to capture dependencies and contextual information.

Feed-Forward Network: The output from the self-attention mechanism is passed through a position-wise feed-forward network.

Layer Normalization and Residual Connections: Layer normalization and residual connections are applied.

3. Decoder Process

Input Embedding and Positional Encoding: The partially generated output sequence is tokenized and embedded with positional encodings added.

Masked Self-Attention Mechanism: The decoder uses masked self-attention to prevent attending to future tokens ensuring that the model generates the sequence step-by-step.

Encoder-Decoder Attention Mechanism: The decoder attends to the encoder's output allowing it to focus on relevant parts of the input sequence.

Feed-Forward Network: Similar to the encoder the output from the attention mechanisms is passed through a position-wise feed-forward network.

Layer Normalization and Residual Connections: Similar to the encoder Layer normalization and residual connections are applied.

4. Training and Inference

Transformers are trained using supervised learning where the model learns to predict the next token in a sequence given the previous tokens.

Transformers have transformed deep learning by using self-attention mechanisms to efficiently process and generate sequences capturing long-range dependencies and contextual relationships. Their encoder-decoder architecture combined with multi-head attention and feed-forward networks enables highly effective handling of sequential data.

Architecture Components:

A standard Transformer has:

1. **Input Embeddings**

- Converts each word/token into a vector.
- Adds positional encoding to maintain word order.

2. **Encoder Block (repeats N times)**

- Self-Attention Layer
- Feed Forward Neural Network
- Residual connection + Layer Normalization

3. Decoder Block (for generation tasks)

- Masked Self-Attention
- Encoder-Decoder Attention
- Feed Forward Layer

✅ 4. Final Linear & Softmax Layer

Converts decoder output into vocabulary probabilities.

Attention Mechanism: Working Principle

1. What is Attention?

Attention allows the model to focus on different parts of the input sequence when producing each word in the output sequence.

Instead of treating all inputs equally, attention gives weights to words based on their relevance to a specific task.

Self-Attention (used in Transformers)

For each token:

- Compare it with every other token in the sequence.
- Calculate attention scores (i.e., how much focus to place on each word).
- Weighted sum of values gives the final representation.

$$Attention(Q,K,V) = softmax\left(\frac{Q_K^T}{\sqrt{d_k}}\right)v$$

Where:

Q = Query

K = Key

V = Value

d _k= Dimension of the key vector (for scaling)

Attention:

Attention is a mechanism that helps the model decide which parts of the input to focus on when performing a task like translation or summarization.

Self-Attention:

Self-Attention is a specific type of attention where a sequence attends to itself. It helps capture relationships within the same sequence — both forwards and backwards.

Attention vs Self-Attention: Key Differences

Feature	Attention	Self-Attention
Input Source	Queries come from decoder, Keys/Values from encoder	Queries, Keys, and Values are from the same input
Usage	Used in encoder-decoder models (e.g., translation)	Used inside encoder or decoder (e.g., BERT, GPT)
Type	Cross-Attention	Intra-sequence Attention
Purpose	Helps decoder focus on relevant input parts	Helps model understand word relationships within input

In Transformers:

- Encoder uses Self-Attention
- Decoder uses:
    - Masked Self-Attention
    - Cross-Attention (Attention to encoder outputs)

Transformer Model – Interview Questions & Answers

1. What is the Transformer model?

The Transformer is a deep learning architecture introduced in 2017 (Vaswani et al.) that relies entirely on attention mechanisms, discarding recurrence and convolutions. It enables parallel processing of sequences and is widely used in NLP tasks.

2. What are the main components of a Transformer model?

- Input Embedding + Positional Encoding
- Encoder (stacked layers): Self-Attention + Feed Forward
- Decoder: Masked Self-Attention + Cross-Attention + Feed Forward
- Output Layer: Linear + Softmax

3. Why are Transformers better than RNNs or LSTMs?

- Parallelizable (faster training)
- Better long-range dependency capture
- More scalable
- Supports bidirectional context (via models like BERT)

4. What are Positional Encodings?

Since Transformers process the input in parallel and not sequentially, positional encodings add information about the position/order of words in the sequence using sine and cosine functions.

5. What is the role of Multi-Head Attention in Transformers?

Multi-head attention allows the model to attend to information from different representation subspaces at different positions. It improves the ability to learn richer representations.

6. How does the decoder differ from the encoder in a Transformer?

The decoder includes:

- Masked Self-Attention (prevents access to future tokens)
- Cross-Attention (attends to encoder outputs)
- While the encoder only uses self-attention.

Self-Attention – Interview Questions & Answers

7. What is Self-Attention?

Self-attention allows a word to attend to all words (including itself) in the same sequence to gather contextual information.

8. How does Self-Attention work mathematically?

For each token, calculate:

$$Attention(Q, K, V) = softmax\left(\frac{Q_K^T}{\sqrt{d_k}}\right) v$$

Where ,

Q,K,V: Projections of the same input

D_k : Dimension of key vectors

9. Why is the dot product divided by $\sqrt{d_k}$?

To prevent extremely large values in the dot product that would make the softmax function saturate, leading to very small gradients.

10. What is Masked Self-Attention?

Masked self-attention prevents the model from "looking ahead" at future tokens in the decoder during training (used in GPT-like models and sequence generation).

Attention Mechanism – Interview Questions & Answers

11. What is the attention mechanism in machine learning?

Attention allows the model to focus more on relevant parts of the input sequence when processing data, enhancing contextual understanding.

12. What is the difference between Attention and Self-Attention?

Aspect	Attention	Self-Attention
Inputs	Query ≠ Key = Value	Query = Key = Value
Used In	Encoder-Decoder (cross)	Encoder & Decoder

Purpose	Align source and target	Capture internal relationships

Query ≠ Key = Value:

- Query comes from the decoder (what we're generating)
- Key and Value come from the encoder (what we're conditioning on)

This allows the decoder to focus on relevant parts of the input sequence.

Query = Key = Value:

- All three vectors come from the same sequence
- Each token looks at all other tokens (including itself) in the same sequence

13. What is Cross-Attention in Transformers?

It is used in the decoder where queries come from the decoder input and keys/values come from the encoder output. It enables the decoder to focus on relevant encoder representations.

14. What is Multi-Head Attention, and why is it used?

Multi-head attention performs attention multiple times in parallel with different learnable linear projections. It allows the model to capture different types of relationships (syntax, semantics, etc.).

15. How do attention weights work?

They determine how much importance to give to each token in the input. The softmax operation ensures these weights sum to 1.

Advanced & Application-Based Questions

16. What is BERT, and how is it related to Transformers?

BERT (Bidirectional Encoder Representations from Transformers) uses only the encoder part of the Transformer for tasks like classification, QA, etc., with masked language modeling and next sentence prediction.

17. What is GPT, and how does it differ from BERT?

GPT (Generative Pre-trained Transformer) uses only the decoder and is trained for autoregressive generation tasks. Unlike BERT, it is unidirectional and optimized for generation.

18. Explain the flow of information in a Transformer encoder.

Input → Embedding + Positional Encoding → Self-Attention → Add & Norm → Feed Forward → Add & Norm → Output → (next encoder layer)

19. What happens if we remove positional encodings from the Transformer?

The model will lose the ability to understand the order of tokens, degrading performance especially in tasks requiring sequence awareness.

20. Can Transformers be used for images or audio?

Yes. Vision Transformers (ViT) and Audio Transformers process patches or chunks like tokens and apply the same attention mechanism, proving effective in non-text domains.

Top SE Block Interview Questions & Answers

1. What is a Squeeze-and-Excitation (SE) Block?

Answer:

An SE Block is a lightweight architectural unit introduced to improve the representational capacity of a neural network by explicitly modeling the interdependencies between channels. It works by using global average pooling to “squeeze” spatial information and then applying a gating mechanism (with two dense layers and a sigmoid function) to “excite” and re-weight each channel.

2. Why is the SE Block used in CNNs?

Answer:

The SE Block helps CNNs focus on the most important feature channels by dynamically adjusting channel-wise feature weights. This improves performance by enhancing meaningful features and suppressing irrelevant ones, leading to better accuracy in tasks like classification and segmentation.

3. What are the main components of an SE Block?

Answer:

- Squeeze: Global Average Pooling to summarize spatial information.
- Excitation: Fully Connected layers (with bottleneck and sigmoid) to generate weights.
- Recalibration: Multiplying the original feature maps by these learned weights.

4. How does the SE Block affect model size and computation?

Answer:

SE Blocks are lightweight and add only a small number of parameters and computations, especially if a bottleneck (reduction ratio) is used in the dense layers. They provide significant performance improvements with minimal overhead.

5. In what types of architectures can SE Blocks be used?

Answer:

- SE Blocks are flexible and can be integrated into:
- ResNet → SE-ResNet
- U-Net → U-Net + SE Blocks
- MobileNet, DenseNet, and many others

They are used in both classification and segmentation networks.

6. How do SE Blocks help in image segmentation tasks like U-Net?

Answer:

In image segmentation tasks, SE blocks help the model focus on more relevant features like organ boundaries or diseased areas by adaptively recalibrating channel importance. This improves precision and recall in pixel-wise predictions.

7. What is the "reduction ratio" in SE Blocks?

Answer:

The reduction ratio (commonly 16 or 8) is a hyperparameter used to reduce the dimensionality in the first fully connected layer of the excitation module. It controls the bottleneck level and affects the trade-off between complexity and performance.

8. Can SE Blocks be applied to 1D or 3D CNNs?

Answer:

Yes. SE Blocks are architecture-agnostic and can be extended to 1D (for sequences) and 3D (for volumes, e.g., medical imaging) by adjusting the global pooling operation accordingly.

10. Global Average Pooling (GAP) in Squeeze-and-Excitation (SE) Block

In an SE Block, Global Average Pooling is used in the Squeeze step to convert each channel's spatial feature map into a single scalar value — essentially summarizing the global information of each channel

Why Global Average Pooling is used:

- Reduces dimensions: Collapses the spatial dimensions (H×W) to a single value per channel.
- Retains channel-wise global context: Helps the model understand which channels are important overall, regardless of location.
- Lightweight: No parameters are added during pooling.
- Prepares for excitation: The vector z is then fed into two dense layers (excitation) to compute channel-wise importance.

Global Average Pooling in SE blocks compresses each channel's spatial information into a global descriptor that tells the model how important that channel is for the final prediction.

In Squeeze-and-Excitation (SE) Block, Global Average Pooling (GAP) produces one number per channel. GAP is designed to summarize the global spatial information of a feature map into a single value — this helps the model learn channel-wise importance during the Excitation phase.

Example:

Let's say your input tensor to the SE block has the shape: $\boldsymbol{X\ ER^{HXWXC} = 32 \times 32 \times 64}$

This means:

32×32 = spatial dimensions (height × width)

64 = number of channels

After Global Average Pooling:

For each of the 64 channels, GAP computes the average of all 32×32 = 1024 values.

So, the output becomes: $\boldsymbol{z \in R = 64}$

a 1D vector of 64 scalar values, where each value is the average activation of one channel.

9. How do you measure the performance of an LLM?

Researchers and practitioners have developed numerous evaluation metrics to gauge the performance of an LLM. Common metrics include:

- Perplexity: Measures how well the model predicts a sample, commonly used in language modeling tasks.
- Accuracy: Used for tasks like text classification to measure the proportion of correct predictions.
- F1 Score: A harmonic mean of precision and recall, used for tasks like named entity recognition.
- **BLEU (Bilingual Evaluation Understudy) score:** Measures the quality of machine generated text against reference translations, commonly used in machine translation.
- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** A set of metrics that evaluate the overlap between generated text and reference text, often used in summarization tasks. They help quantify the model's effectiveness and guide further improvements.